10

Sharing data volumes between machines: EFS

This chapter covers

- Creating a highly available shared filesystem
- Mounting your shared filesystem on multiple EC2 instances
- Sharing files between EC2 instances
- Testing performance of your shared filesystem
- Backing up your shared filesystem

Many legacy applications store state in files on disk. Therefore, using Amazon S3, an object store, as described in chapter 8 is not possible by default. Using block storage as discussed in the previous chapter might be an option, but does not allow you to access files from multiple machines in parallel. Hence you need a way to share the files between virtual machines. With Elastic File System (EFS), you can share data between multiple EC2 instances and your data is replicated between multiple *availability zones* (AZ).

EFS is based on the NFSv4.1 protocol, so you can mount it like any other filesystem. In this chapter you learn how to set up EFS, tweak performance, and back your data.

EFS ONLY WORKS WITH LINUX At this time, EFS is not supported by Windows EC2 instances.

Examples are 100% covered by the Free Tier

The examples in this chapter are completely covered by the Free Tier. As long as you don't run the examples longer than a few days, you won't pay anything. Keep in mind that this only applies if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

Let's take a closer look at how EFS works compared to Elastic Block Store (EBS) and the instance store, which were introduced in the previous chapter. An EBS volume is tied to a data center (also called an AZ) and can only be attached over the network to a single EC2 instance from the same data center. Typically EBS volumes are used as the root volumes that contain the operating system, or for relational database systems to store the state. An instance store consists of a hard drive directly attached to the hardware the VM is running on. An instance store can be regarded as ephemeral storage, and is therefore used for caching or for NoSQL databases with embedded data replication only. In contrast, the EFS filesystem can be used by multiple EC2 instances from different data centers in parallel. Additionally, the data on the EFS filesystem is replicated among multiple data centers and remains available even if a whole data center suffers from an outage, which is not true for EBS and instance stores. Figure 10.1 shows the differences.

Now let's take a closer look at EFS. There are two main components to know about:

1. *Filesystem*—Stores your data
2. *Mount target*—Makes your data accessible

The filesystem is the resource that stores your data in an AWS region, but you can't access it directly. To do so, you must create an EFS mount target in a subnet. The mount target provides a network endpoint that you can use to mount the filesystem on an EC2 instance via NFSv4.1. The EC2 instance must be in the same subnet as the EFS mount target, but you can create mount targets in multiple subnets. Figure 10.2 demonstrates how to access the filesystem from EC2 instances running in multiple subnets.

Equipped with the EFS theory about filesystems and mount targets, you can now apply your knowledge to solve a real problem.

Linux is a multiuser operating system. Many users can store data and run programs isolated from each other. Each user can have a home directory that usually is stored

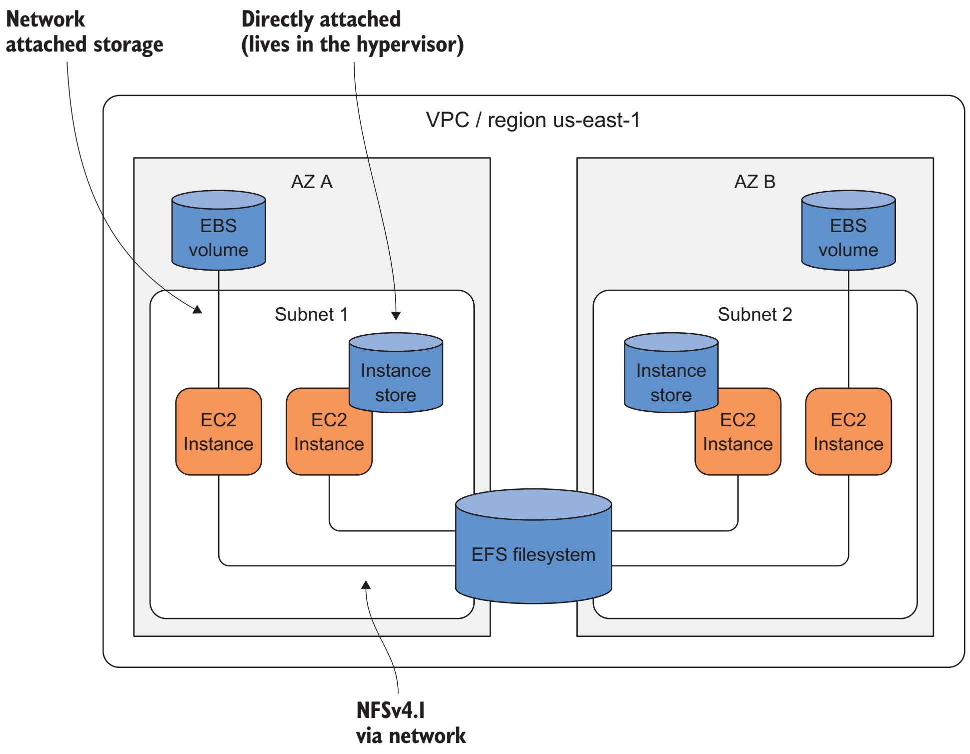


Figure 10.1 Comparing EBS, instance stores, and EFS

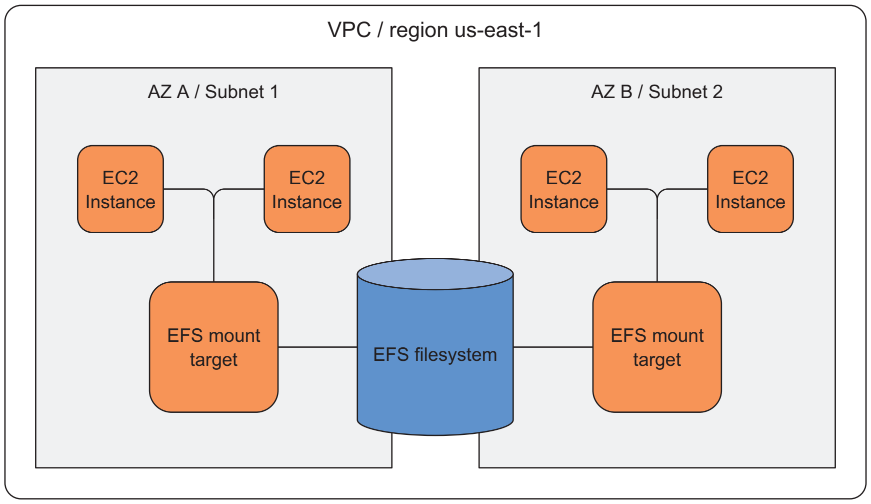


Figure 10.2 Mount targets provide an endpoint for EC2 instances to mount the filesystem in a subnet.

under /home/$username. If the user name is michael, the home directory would be /home/ michael, and only that user would be allowed to read and write in /home/ michael. The `ls -d -l /home/*` command lists all home directories.

```
$  ls -d -l /home/*                                   <-- List all home directories with absolute paths.
drwx------ 2 andreas    andreas    4096 Jul 24 06:25 /home/andreas    <-- /home/andreas can only be accessed by the user and group andreas.
drwx------ 3 michael    michael    4096 Jul 24 06:38 /home/michael    <-- /home/michael can only be accessed by the user and group michael.
```

If you are using multiple EC2 instances, your users will have a separate home folder on each EC2 instance. If a Linux user uploads a file on one EC2 instance, they can't access the file on another EC2 instance. To solve this problem, create a filesystem and mount EFS on each EC2 instance under /home. The home directories are then shared across all your EC2 instances, and users will feel at home no matter which VM they log in to. In the following sections, you will build this solution step-by-step. First, you will create the filesystem.

10.1 Creating a filesystem

The filesystem is the resource that stores your files, directories, and links. Like S3, EFS grows with your storage needs. You don't have to provision the storage up front. The filesystem is located in an AWS region and replicates your data under the covers across multiple availability zones. You will use CloudFormation to set up the filesystem now.

10.1.1 Using CloudFormation to describe a filesystem

The bare minimum to describe a filesystem resource is shown here.

Listing 10.1 CloudFormation snippet of an EFS filesystem resource

```
Resources:                                <-- Specifies the stack resources and their properties
  [...]
  FileSystem:
    Type: 'AWS::EFS::FileSystem'
    Properties: {}                        <-- Nothing needs to be configured.
```

Optionally, you can add tags to track costs or add other useful meta data with the `FileSystemTags` property.

10.1.2 Pricing

Calculating EFS costs is simple. You only need to know the amount of storage you use in GB. EFS charges you per GB per month. If your EFS filesystem is 5 GB in size, you will be charged 5 GB x 0.30 (USD/GB/month) in us-east-1, for a total of $1.50 USD a month. Please check https://aws.amazon.com/efs/pricing to get the latest pricing information for your region. The first 5 GB per month are free in the first year of your AWS account (Free Tier).

The filesystem is now described in CloudFormation. To use it, you need to create at least one mount point. Creating a mount target is the subject of the next section.

10.2 Creating a mount target

An EFS mount target makes your data available to EC2 instances via the NFSv4.1 protocol in a single AZ. The EC2 instance communicates with the mount target via a TCP/IP network connection. As you learned in section 6.4, security groups are how you control network traffic on AWS. You can use a security group to allow inbound traffic to an EC2 instance or an RDS database, and the same is true for a mount target. Security groups control which traffic is allowed to enter the mount target. The NFS protocol uses port 2049 for inbound communication. Figure 10.3 shows how mount targets are protected.

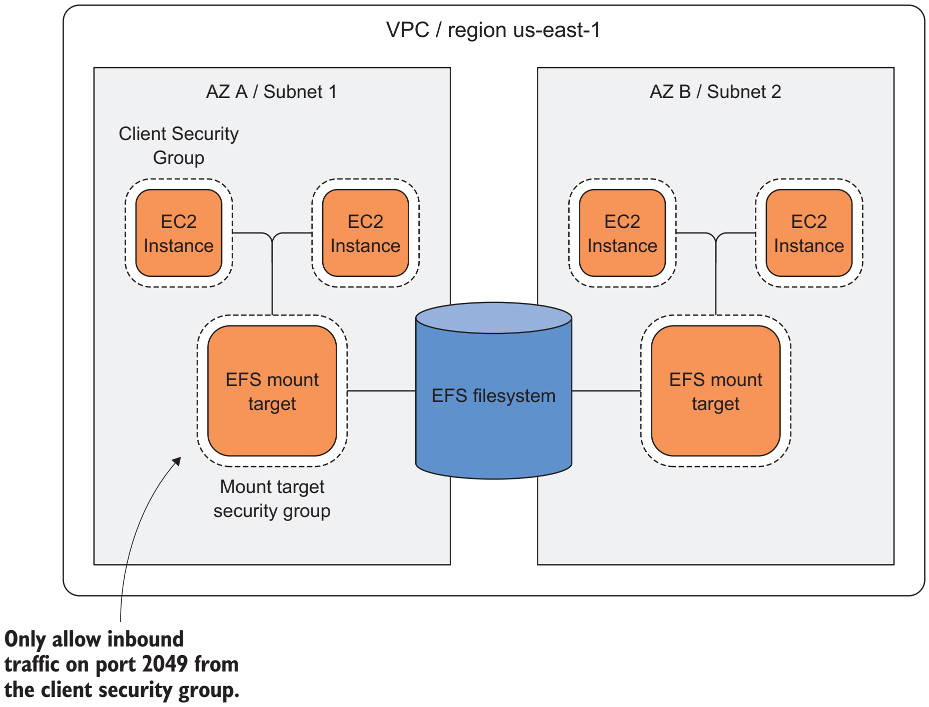


Figure 10.3 EFS mount targets are protected by security groups.

In our example, to control traffic as tightly as possible, you won't white list IP addresses. Instead, you'll create two security groups. The client security group will be attached to all EC2 instances that want to mount the filesystem. The mount target security group allows inbound traffic on port 2049 only for traffic that comes from the client security group. This way, you can have a dynamic fleet of clients who are allowed to send traffic to the mount targets. You used the same approach for the SSH bastion host in section 6.4.

You can use CloudFormation to manage an EFS mount target. The mount target references the filesystem, needs to be linked to a subnet, and is also protected by at least one security group. You will first describe the security groups, followed by the mount target, as shown in listing 10.2.

Listing 10.2 CloudFormation snippet of an EFS mount target and security groups

```
Resources:
  [...]
  EFSClientSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'EFS Mount target client'
      VpcId: !Ref VPC
  MountTargetSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'EFS Mount target'
      SecurityGroupIngress:
      - FromPort: 2049
        IpProtocol: tcp
        SourceSecurityGroupId: !Ref EFSClientSecurityGroup
        ToPort: 2049
      VpcId: !Ref VPC
  MountTargetA:
    Type: 'AWS::EFS::MountTarget'
    Properties:
      FileSystemId: !Ref FileSystem
      SecurityGroups:
      - !Ref MountTargetSecurityGroup
      SubnetId: !Ref SubnetA
```

The client security group needs no rules. It's just used to mark traffic.

Allow traffic on port 2049.

Only allow traffic from the client security group.

Connect mount target with the filesystem.

Assign the security group.

Connect with a subnet which also determines the AZ.

Copy the MountTargetA resource and also create a mount target for SubnetB.

```
Resources:
  [...]
  MountTargetB:
    Type: 'AWS::EFS::MountTarget'
    Properties:
      FileSystemId: !Ref FileSystem
      SecurityGroups:
      - !Ref MountTargetSecurityGroup
      SubnetId: !Ref SubnetB
```

The other subnet is used.

The mount targets can now be used in the next section, where you finally mount the /home directory.

10.3 Mounting the EFS share on EC2 instances

EFS creates a DNS name for each filesystem following the schema *$FileSystemID*.efs.*$Region*.amazonaws.com. Inside an EC2 instance, this name resolves to the mount target of the instance's AZ. AWS suggests the following mount options:

- `nfsvers=4.1`—Specifies which version of the NFS protocol to use.
- `rsize=1048576`—Read data block size, in bytes, to be transferred at one time.
- `wsize=1048576`—Write data block size, in bytes, to be transferred at one time.
- `hard`—If the EFS share is down, wait for the share to come back online.
- `timeo=600`—The time in deciseconds (tenths of a second) the NFS client waits for a response before it retries an NFS request.
- `retrans=2`—The number of times the NFS client retries a request before it attempts further recovery action.

This snippet shows the full mount command:

```
$ mount -t nfs4 -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,\
➥ retrans=2 $FileSystemID.efs.$Region.amazonaws.com:/ $EFSMountPoint
```

Replace *`$FileSystemID`* with the EFS filesystem, such as fs-123456. Replace *`$Region`* with the region, such as us-east-1, and *`$EFSMountPoint`* with the local path where the filesystem is mounted. You can also use the /ets/fstab file to automatically mount on startup:

```
$FileSystemID.efs.$Region.amazonaws.com:/ $EFSMountPoint nfs4 nfsvers=4.1,
➥ rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,_netdev 0 0
```

To make sure that the DNS name can be resolved and the other side is listening to the port, you can use the following Bash script to wait until the mount target is ready:

```
$ while ! nc -z $FileSystemID.efs.$Region.amazonaws.com 2049;
➥ do sleep 10; done
$ sleep 10
```

To use the filesystem, you have to mount it on EC2 instances using one of the mount targets that you created. It's now time to add two EC2 instances to the CloudFormation template. Each EC2 instance should be placed in a different subnet and mount the filesystem to /home. The /home directory will exist on both EC2 instances, and it will also contain some data (such as the folder ec2-user). You have to ensure that you're copying the original data the first time before you mount the EFS filesystem, which is empty by default. This listing describes the EC2 instance that copies the existing /home folder before the shared home folder is mounted.

Listing 10.3 An EC2 instance in SubnetA and security group resources

```
Resources:
  [...]
  EC2SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'EC2 instance'
      SecurityGroupIngress:
      - CidrIp: '0.0.0.0/0'
        FromPort: 22
        IpProtocol: tcp
        ToPort: 22
      VpcId: !Ref VPC
  EC2InstanceA:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-6057e21a'
      InstanceType: 't2.micro'
      KeyName: mykey
      NetworkInterfaces:
      - AssociatePublicIpAddress: true
        DeleteOnTermination: true
        DeviceIndex: 0
        GroupSet:
        - !Ref EC2SecurityGroup
        - !Ref EFSClientSecurityGroup
        SubnetId: !Ref SubnetA
      UserData:
        'Fn::Base64': !Sub |
          #!/bin/bash -x
          bash -ex << "TRY"
            while ! nc -z ${FileSystem}.efs.${AWS::Region}.amazonaws.com 2049;
➥ do sleep 10; done
            sleep 10
            mkdir /oldhome
            cp -a /home/. /oldhome
            echo "${FileSystem}.efs.${AWS::Region}.amazonaws.com:/ /home nfs4
➥ nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,
➥retrans=2,_netdev 0 0" >> /etc/fstab
            mount -a
            cp -a /oldhome/. /home
          TRY
          /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName}
➥ --resource EC2InstanceA --region ${AWS::Region}
    CreationPolicy:
      ResourceSignal:
        Timeout: PT10M
    DependsOn:
    - VPCGatewayAttachment
    - MountTargetA
```

Security Group to allow SSH traffic from the internet

Ensure the EC2 instance gets a public IP address for SSH access.

Attach the client mount target security group.

Attach the security group to allow SSH.

Place instance into subnet A.

Wait until filesystem is available.

Create temporary folder for /home content.

Copy existing /home to /oldhome and preserve permissions (-a).

Mount filesystem.

Copy /oldhome to new /home.

Let CloudFormation know that the EC2 instance resource is efs-with-backup.

Wait for the internet gateway.

Wait for the mount target.

After the EC2 instance is launched, you will find the first data on the EFS share. The second EC2 instance is similar but in a different subnet, and does not copy the existing

/home content because this was already done by the previous EC2 instance. Here are the details.

Listing 10.4 An EC2 instance in SubnetB and security group resources

```
Resources:
  [...]
  EC2InstanceB:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-6057e21a'
      InstanceType: 't2.micro'
      KeyName: mykey
      NetworkInterfaces:
      - AssociatePublicIpAddress: true
        DeleteOnTermination: true
        DeviceIndex: 0
        GroupSet:
        - !Ref EC2SecurityGroup
        - !Ref EFSClientSecurityGroup
        SubnetId: !Ref SubnetB          <- Place into the other subnet.
      UserData:
        'Fn::Base64': !Sub |
          #!/bin/bash -x
          bash -ex << "TRY"
            while ! nc -z ${FileSystem}.efs.${AWS::Region}.amazonaws.com 2049;
➥ do sleep 10; done
            sleep 10
            echo "${FileSystem}.efs.${AWS::Region}.amazonaws.com:/ /home nfs4
➥ nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,
➥ retrans=2,_netdev 0 0" >> /etc/fstab
            mount -a
          TRY
          /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName}
➥ --resource EC2InstanceB --region ${AWS::Region}
    CreationPolicy:
      ResourceSignal:
        Timeout: PT10M
    DependsOn:
    - VPCGatewayAttachment
    - MountTargetB
```

The old /home is not copied here. This is already done on the first EC2 instance in subnet A.

To make things easier, you can also add outputs to the template to expose the public IP addresses of your EC2 instances like this:

```
Outputs:
  EC2InstanceAIPAddress:
    Value: !GetAtt 'EC2InstanceA.PublicIp'
    Description: 'EC2 Instance (AZ A) public IP address (connect via SSH)'
  EC2InstanceBIPAddress:
    Value: !GetAtt 'EC2InstanceB.PublicIp'
    Description: 'EC2 Instance (AZ B) public IP address (connect via SSH)'
```

The CloudFormation template is now complete. It contains:

- Filesystem (EFS)
- Two EFS mount targets in SubnetA and SubnetB
- Security groups to control traffic to the mount targets
- EC2 instances in both subnets, including a `UserData` script to mount the filesystem

It's now time to create a stack based on your template, to create all the resources in your AWS account. You can find the full code for the template at /chapter10/template.yaml in the book's code folder. You can use the AWS CLI to create the stack:

```
$ aws cloudformation create-stack --stack-name efs \
➥ --template-url https://s3.amazonaws.com/awsinaction-code2/\
➥ chapter10/template.yaml
```

Where is the template located?

You can find the template on GitHub. You can download a snapshot of the repository at https://github.com/AWSinAction/code2/archive/master.zip. The file we're talking about is located at chapter10/template.yaml. On S3, the same file is located at http://mng.bz/XIUE.

Once the stack is in the state CREATE_COMPLETE, you have two EC2 instances running. Both mounted the EFS share to /home. You also copied the old /home data to the EFS share. It's time to connect to the instances via SSH and do some tests (in the next section), to see if users can really share files between the EC2 instances in their home directory.

10.4 Sharing files between EC2 instances

Open an SSH connection to the virtual machine in subnet A. You can use the AWS CLI to get the stack output, from which you can get the public IP address:

```
$ aws cloudformation describe-stacks --stack-name efs \
➥ --query "Stacks[0].Outputs"
[{
  "Description": "[...]",
  "OutputKey": "EC2InstanceAIPAddress",
  "OutputValue": "54.158.102.196"
}, {
  "Description": "[...]",
  "OutputKey": "EC2InstanceBIPAddress",
  "OutputValue": "34.205.4.174"
}]
```

Use the SSH key `mykey` to authenticate, and replace *`$PublicIpAddress`* with the IP address of the `EC2InstanceAIPAddress` output from the stack:

```
$ ssh -i $PathToKey/mykey.pem ec2-user@$PublicIpAddress
```

Open a second SSH connection to the virtual machine in subnet B. Use the same command you just used, but this time, replace *`$PublicIpAddress`* with the IP address of the `EC2InstanceBIPAddress` output from the stack.

You now have two SSH connections open. In both SSH sessions you should be located in the /home/ec2-user folder. Check if this is true on both machines:

```
$ pwd
/home/ec2-user          <-- The output confirms that your
                            current directory is /home/ec2-user.
```

Also check if there are any files or folders in /home/ec2-user:

```
$ ls                    <-- If no data is returned, the folder
                            /home/ec2-user is empty.
```

Now, create a file on one of the machines:

```
$ touch i-was-here         <-- touch creates an empty file.
```

On the other machine, confirm that you can see the new file:

```
$ ls
i-was-here                 <-- You can see the file.
```

You now have access to the same home directory on both machines. You could add hundreds of machines to this example. All would share the same home directory, and your users would be able to access the same home directory on all EC2 instances. You can apply the same mechanism to share files between a fleet of web servers (for example, the `/var/www/html` folder), or to design a highly available Jenkins server (such as `/var/lib/jenkins`).

To operate the solution successfully, you also need to take care of backups, performance tuning, and monitoring. You will learn about this in the following sections.

10.5 Tweaking performance

To compare EFS with other storage options, we'll use the same simple performance test that we used in section 9.1.3 to test the performance of an EBS volume. The tool `dd` can perform block-level reads and writes.

```
$ sudo dd if=/dev/zero of=/home/ec2-user/tempfile bs=1M count=1024 \   <-- Writes 1 MB 1,024 times
➥ conv=fdatasync,notrunc
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 10.4138 s, 103 MB/s   <-- 103 MB/s write performance

$ echo 3 | sudo tee /proc/sys/vm/drop_caches     <-- Flushes caches
3

$ sudo dd if=/home/ec2-user/tempfile of=/dev/null bs=1M count=1024    <-- Reads 1 MB 1,024 times
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 10.2916 s, 104 MB/s    <-- 104MB/s read performance
```

Please keep in mind that this performance tests assumes files of 1 MB. Depending on your workload, files can be smaller or bigger, which leads to different results. Usually, throughput will degrade for smaller files. Comparing these numbers to the `dd` results on EBS and instance store is only of limited informative value.

10.5.1 Performance mode

So far, you've used the General Purpose performance mode, which is fine for most workloads, especially latency-sensitive ones where small files are served most of the time. The `/home` directory is a perfect example of such a workload. Users are not constantly opening and saving files. Instead they list files from time to time and then open a specific file. When the user opens a file, they expect low latency to get the file instantaneously.

But sometimes, EFS is used to store massive amounts of data for analytics. For data analytics, latency is not important. Throughput is the metric you want to optimize instead. If you want to analyze gigabytes or terabytes of data, it doesn't matter if your time to first byte takes 1 ms or 100 ms. Even a small increase in throughput will decrease the time it will take to analyze the data. For example, analyzing 1 TB of data with 100 MB/sec throughput will take 174 minutes. That's almost three hours, so the first few milliseconds don't really matter. Optimizing for throughput can be achieved using the Max I/O performance mode. The performance mode being used by an EFS filesystem cannot be changed—you set it when the filesystem is created. Therefore, to change the performance mode, you have to create a new filesystem. We recommend you start with the General Purpose performance mode if you are unsure which mode fits best for your workload. You will learn how to check whether you made the right decision by looking at monitoring data in the following section.

10.5.2 Expected throughput

The performance of EFS scales with the amount of storage you use. EFS also allows for bursting, because many workloads require high performance for a short period of time and are idle for most of the rest of the time.

The baseline rate is 51.2 KB/s per 1.1 GB of storage (or 52.4 MB/s per 1100 GB). You can burst 50% of the time if the filesystem is not accessed for the remaining time. Table 10.1 shows how you can calculate your burst rate.

Table 10.1 EFS burst rate

filesystem size	Burst rate
< 1100 GB	104.9 MB/s
>= 1100 GB	104.9 MB/s per 1100 GB of data stored

The burst rate is exactly the rate we measured in the simple performance test at the beginning of this section! To get the exact numbers for how long you can burst, you can consult the official documentation: “Throughput Scaling in Amazon EFS” at http://mng.bz/Y5Q9.

The throughput rules are complicated. Luckily, you can use CloudWatch to observe the numbers and send an alert if you run out of credits. That’s the topic of the next section.

10.6 Monitoring a filesystem

CloudWatch is the AWS service that stores all kinds of metrics. EFS sends useful metrics, the most important of which to watch are:

- `BurstCreditBalance`—The current credit balance. You need credits to be able to burst throughput to your EFS filesystem.
- `PermittedThroughput`—The throughput you have at this moment. Takes burst credits and size into account.
- `Read/Write/Metadata/TotalIOBytes`—Information in bytes about reads, writes, metadata, and total I/O usage.
- `PercentIOLimit`—How close you are to hitting the I/O limit. If this metric is at 100%, the Max I/O performance mode would be a good choice.

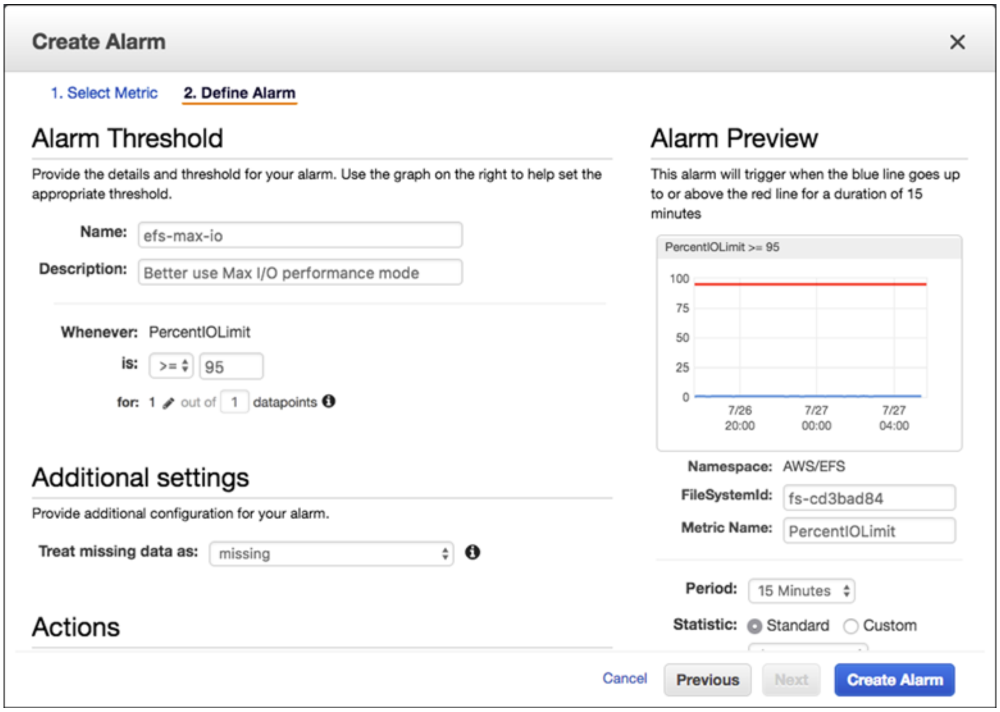
Let’s look at those metrics in more detail now. You will also get some hints about useful thresholds to define alarms on those metrics.

10.6.1 Should you use Max I/O Performance mode?

The `PercentIOLimit` metric shows how close a filesystem is to reaching the I/O limit of the General Purpose performance mode (not used for other modes). If this metric is at 100% for more than half of the time, you should consider creating a new filesystem with Max I/O performance mode enabled. It makes sense to create an alarm on this metric, as shown in figure 10.4. To create a CloudWatch Alarm in the Management Console:

1. Open the CloudWatch Management Console: https://us-east-1.console.aws.amazon.com/cloudwatch/home.
2. Click the Alarms link on the left.
3. Click the Create Alarm button.
4. Under EFS Metrics, click File System Metrics.
5. Select the PercentIOLimit metric.
6. Click the Next button.
7. Fill out the fields as shown in figure 10.4.

You might set an alarm to trigger if the 15-minute average of the metric is higher than 95% for 4 out of 4 datapoints. The alarm action usually sends a message to an SNS topic which you can subscribe to via email.


Create Alarm

1. Select Metric   2. Define Alarm

Alarm Threshold

Provide the details and threshold for your alarm. Use the graph on the right to help set the appropriate threshold.

Name: efs-max-io

Description: Better use Max I/O performance mode

Whenever: PercentIOLimit

is: >= 95

for: 1 out of 1 datapoints

Additional settings

Provide additional configuration for your alarm.

Treat missing data as: missing

Actions

Alarm Preview

This alarm will trigger when the blue line goes up to or above the red line for a duration of 15 minutes

Namespace: AWS/EFS

FileSystemId: fs-cd3bad84

Metric Name: PercentIOLimit

Period: 15 Minutes

Statistic: [✓ Standard] Custom

Figure 10.4 Creating a CloudWatch Alarm on EFS's PercentIOLimit metric

10.6.2 Monitoring your permitted throughput

In the previous section, we promised that there is an easier way to get access to the actual throughput of your filesystem besides doing the math. The `PermittedThroughput` metric provides this important information. It takes into account the size of your filesystem and your credit balance to calculate the permitted throughput of your filesystem. Since your credit balance changes all the time (you either consume credits or new credits are added), the permitted throughput can be volatile. Figure 10.5 shows a line chart of the `PermittedThroughput` and the `BurstCreditBalance` at the moment when you run out of credits.

If you rely on credits to get the expected performance, you should create an alarm that monitors your `BurstCreditBalance`. You might set an alarm to trigger if the 10-minute average of the `BurstCreditBalance` metric is lower than 192 GB for one consecutive period, which is the last hour where you can burst at 100 MB/sec. Don't set the threshold too low: you need some time to react! (You can add dummy files to increase the EFS filesystem size, which increases throughput.)

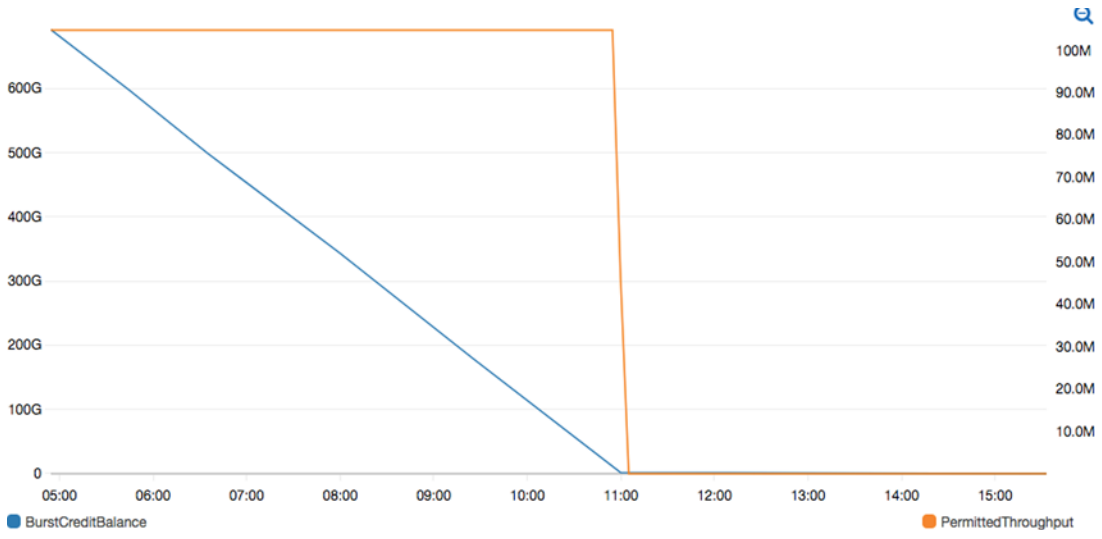


Figure 10.5 Creating a CloudWatch Alarm on EFS's `PermittedThroughout` metric

Listing 10.5 shows a CloudFormation snippet for an alarm to monitor the `BurstCreditBalance`.

Listing 10.5 Alarm resource on the `BurstCreditBalance` metric

```
Resources:
  [...]
  FileSystemBurstCreditBalanceTooLowAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmDescription: 'EFS file system is running out of burst credits.'
      Namespace: 'AWS/EFS'
      MetricName: BurstCreditBalance          <- Name of the metric
      Statistic: Average
      Period: 600
      EvaluationPeriods: 1
      ComparisonOperator: LessThanThreshold
      Threshold: 192416666667                 <- 192 GB in bytes (last hour when you can burst at 100 MB/sec)
      AlarmActions:
      - 'arn:aws:sns:us-east-1:123456789012:SNSTopicName'   <- SNS topic ARN to send the alert to. You can also define a SNS topic resource in your template and reference it here.
      Dimensions:
      - Name: FileSystemId
        Value: !Ref FileSystem
```

10.6.3 Monitoring your usage

Access to EFS are either reads, writes, or metadata (metadata is not included in reads or writes). Metadata could be the size and ownership information about a file, or it could be locks to avoid concurrent access to a file. A stacked area chart in CloudWatch can give you a good overview of all the activity. Figure 10.6 shows such a chart.

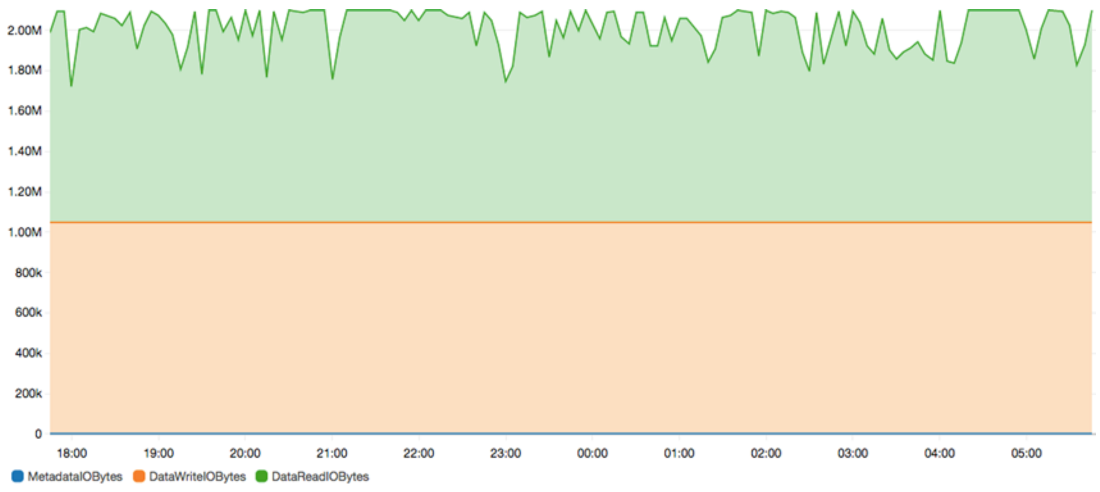


Figure 10.6 CloudWatch graph on EFS usage

Creating an alarm on the usage data makes sense if you know your workload very well. Otherwise you should be safe enough with the alarms from the previous metrics.

10.7 Backing up your data

EFS stores all your files on multiple disks in multiple availability zones. Thus the chances of losing data because of a hardware issue are low. But what about a human error like `rm -rf /`, which removes all the files on Linux (including files in mounted folders)? Or an application bug that corrupts data? Unfortunately, EFS does not provide a native way to back up your EFS filesystem at the time of writing. But there are multiple options to create your own backup solution:

- Sync the files to S3 from time to time.
- Sync the files to an EBS volume from time to time, and create a snapshot of the volume after each sync.
- Sync the files to another EFS filesystem.
- Use a third-party backup solution. EFS is just another volume on your operating system.

If you need a cost effective way to back up the history of changes to EFS, we recommend you use an EBS volume and snapshots. As you learned in section 9.1, EBS snapshots are block-level incremental backups. This means only changes to the blocks are stored when performing multiple snapshots. One disadvantage of this solution is that your data must be small enough to fit on a single EBS volume. If it's not, consider using a second EFS filesystem for your backups.

To implement the EBS backup strategy, you need to add an EBS volume, and attach it to one of the EC2 instances. Finally, you have to implement the logic to synchronize the mounted filesystem with EBS and trigger a snapshot. Let's start by adding the EBS volume.

10.7.1 Using CloudFormation to describe an EBS volume

The EBS volume needs to be defined in the CloudFormation template. You also need to configure that the volume should be attached to the EC2 instance in subnet A. Listing 10.6 shows the CloudFormation snippet.

Listing 10.6 EBS volume resource attached to an EC2 instance

```
Resources:
  [...]
  EBSBackupVolumeA:
    Type: 'AWS::EC2::Volume'
    Properties:
      AvailabilityZone: !Select [0, !GetAZs '']
      Size: 5
      VolumeType: gp2
  EBSBackupVolumeAttachmentA:
    Type: 'AWS::EC2::VolumeAttachment'
    Properties:
      Device: '/dev/xvdf'
      InstanceId: !Ref EC2InstanceA
      VolumeId: !Ref EBSBackupVolumeA
```

The EBS volume needs to be in the same AZ as the EC2 instance.

5 GB in size (you can increase this)

Use the SSD backed general purpose storage type.

The EC2 instance as one side of the volume attachment

The EBS volume as the other side of the volume attachment.

Now, the volume is connected to the EC2 instance in the CloudFormation template.

10.7.2 Using the EBS volume

It might take a moment before the operating system can see the EBS volume as a new disk. A plain new EBS volume is not yet formatted, so you have to format the disk with a filesystem at first use. After that, you can mount the disk. You will use a cron job that runs every 15 minutes to:

1. Copy the files from /home (EFS) to /mnt/backup (EBS) using rsync.
2. Freeze the backup mount, to prevent any additional writes using the `fsfreeze` command.
3. Start the snapshot creation using the AWS CLI.
4. Unfreeze the backup mount using the `fsfreeze -u` command.

The following listing extends the user data script executed by the EC2 instance on startup.

Listing 10.7 Mount the EBS volume and back up the data from EFS periodically

```
[...]
/opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName}
➥ --resource EC2InstanceA --region ${AWS::Region}

while ! [ "`fdisk -l | grep '/dev/xvdf' | wc -l`" -ge "1" ]; do
```

Wait until EBS volume is attached.

```
  sleep 10
done

if [[ "`file -s /dev/xvdf`" != *"ext4"* ]]; then     <-- Format EBS
  mkfs -t ext4 /dev/xvdf                                  volume if needed.
fi

mkdir /mnt/backup
echo "/dev/xvdf /mnt/backup ext4 defaults,nofail 0 2" >> /etc/fstab
mount -a                                   <-- Mount EBS volume.

cat > /etc/cron.d/backup << EOF
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/opt/aws/bin
MAILTO=root
HOME=/
*/15 * * * *
➥ root rsync -av --delete /home/ /mnt/backup/ ;
➥ fsfreeze -f /mnt/backup/ ;
➥ aws --region ${AWS::Region} ec2 create-snapshot
➥ --volume-id ${EBSBackupVolumeA} --description "EFS backup" ;
➥ fsfreeze -u /mnt/backup/
EOF                                        <-- Install backup cron job.
```

To allow the EC2 instance to create a snapshot of its own EBS volume, you have to add an instance profile with a IAM role that allows the `ec2:CreateSnapshot` action. Listing 10.8 shows the CloudFormation snippet.

Listing 10.8 CloudFormation snippet of an EC2 Instance Profile resource

```
Resources:
  [...]
  InstanceProfile:
    Type: 'AWS::IAM::InstanceProfile'
    Properties:
      Roles:
      - !Ref Role
  Role:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Principal:
            Service: 'ec2.amazonaws.com'
          Action: 'sts:AssumeRole'
      Policies:
      - PolicyName: ec2
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
          - Effect: Allow
            Action: 'ec2:CreateSnapshot'
            Resource: '*'
```

Last but not least, you have to attach the instance profile to the EC2 instance by modifying the existing `EC2InstanceA` resource.

```
Resources:
  [...]
  EC2InstanceA:
    Type: 'AWS::EC2::Instance'
    Properties:
      IamInstanceProfile: !Ref InstanceProfile
      [...]
```

It's now time to test the new stack. Don't forget to delete the old stack first. Then, create a new stack based on the extended template. You can find the full code for the template at /chapter10/template-with-backup.yaml in the book's code folder.

```
$ aws cloudformation delete-stack --stack-name efs
$ aws cloudformation create-stack --stack-name efs-with-backup \
➥ --template-url https://s3.amazonaws.com/awsinaction-code2/\
➥ chapter10/template-with-backup.yaml \
➥ --capabilities CAPABILITY_IAM
```

Where is the template located?

You can find the template on GitHub. You can download a snapshot of the repository at https://github.com/AWSinAction/code2/archive/master.zip. The file we're talking about is located at chapter10/template-with-backup.yaml. On S3, the same file is located at http://mng.bz/K87P.

When the stack is in the CREATE_COMPLETE state, you have two EC2 instances running. Every 15 minutes, a new EBS snapshot will be created. You need to be patient to validate that the snapshots are working.

Cleaning up

It's time to delete the running CloudFormation stack:

```
$ aws cloudformation delete-stack --stack-name efs-with-backup
```

Also use the Management Console to delete all EBS snapshots that have been created as periodical backups.

You now have your own backup solution implemented. Keep in mind that the EBS volume does not automatically grow with your EFS filesystem. You have to adjust the size of the EBS volume manually.

Summary

- EFS provides a NFSv4.1-compliant filesystem that can be shared between Linux EC2 instances in different availability zones.
- EFS mount targets are bound to an availability zone and are protected by security groups.
- You need at least two mount targets in different AZs for high availability.
- EFS does not provide snapshots for point-in-time recovery.
- Data that is stored in EFS is replicated across multiple AZs.